Iterative MapReduce on HBase

In this project, like the previous one, Hadoop was setup first. This was a pseudo-distributed setup. This stage also involved the configuration of the properties of the framework in the xml files. After this,HBase was installed and the configurations were modified for the same. As mentioned in the discussion This was done in the xml files as discussed in the discussion.
Based on the data provided, the schema consisted of each data point consisting of 10 properties. These basically where categorized into two column qualifiers “Area” and “Property," each of which had 5 properties each. The ML dataset provided consisted of 768 such data points which would be used for the clustering based on the algorithm implemented.

Load MapReduce Job

Firstly, a MapReduce job was written to put in the data into the input table “Energy”. Here I decided to use ***rowi*** as the key with the values based on the 10 different values as in the datafile.
The first mapper would initially associate a hashcode for a row of the dataset with a point, which in turn would be done for all the 768 points.
Then, using a single reducer, I decided to aggregate this data and input it into the “Energy” table based on the schema given in the HW discussion. A single reducer was used based on the assumption that the dataset would not be too large hence not leading to the single reducer being the bottleneck in this job.
After this, I decided to create the “Center” table based on the initial k value. This k value was taken from the command line as an argument. k points from the “Energy” table were used to insert these values into the “Center” table.
In this way the tables are created. Also, a condition for table’s existence and deletion is done in this part of the code

```
if(admin.tableExists("Energy"))
{
        admin.disableTable("Energy") ;
        admin.deleteTable("Energy") ;
        System.out.println("******Table Energy already exists..disabling and deleting it*******");
}
```

KMeans MapReduce Job

The KMeans consisted of a single MapReduce job which begin with a Map<String,KeyValue[]> centers which would load the values of the “Center” table for computation of the Means for every iteration. This was done using a ***LoadValue*** function in the setup of the Mapper. This setup function is called only once as needed for this application.

Then the function ***NearestCenter*** was used to compute the value of the data points from the “Energy” table to the centroid values in the “Center” table. This would be sent to the Reducer in the context.

In the Reducer, I decided on KMeans.setNumReduceTasks(k). This was needed as we would have all the datapoints grouped into k clusters. Hence, each reducer would receive one of these k sets for computation.
Here too, the same setup function as the Mapper was used to load the values from the "Center" table.
The values in the Result of the Reducer consisted of the datapoints belonging to a particular cluster. These were used to compute the NewMean for these respective points.
Then, the distance between the NewMean and the OldCenter would be calculate to see how much do they vary.
Setting a threshold of 0.01 after testing, the Counter in the context would be incremented only if the distance was greater than 0.01. Here, the new centroid values would also be inserted into center table as well.

This MapReduce stage would be a part of the loop that would run as long as the Counter does not reach a value of zero. This would signify convergence of the KMeans job.

The TableMpaReduceUtil was used for the mapper and reducer jobs which extended TableMapper.

```
TableMapReduceUtil.initTableMapperJob(
                                        "Energy",
                                        scan,
                                        Program.KMeansMapper.class,
                                        Text.class, Result.class, KMeans);
```

Please refer to the source code attached in the zip file submitted for further comments.